TinySec: A Link Layer Security Architecture for Wireless Sensor Networks

Chris Karlof
ckarlof@cs.berkeley.edu
UC Berkeley

Naveen Sastry
nks@cs.berkeley.edu
UC Berkeley

David Wagner
daw@cs.berkeley.edu
UC Berkeley

To be published at Sensys ’04

CS 730 Fall 2004
Presenter: Chien-Liang Fok
What is TinySec?

- A light-weight, generic security software packet for SN
  - Easy to add to existing applications
  - Adds little cost (memory, bandwidth, power)
- Security is necessary for widespread deployment
Paper Contributions

1. Introduce TinySec and its implementation
2. Explore tradeoffs between performance, transparency, and security
3. Measure the cost of TinySec
Challenges Unique to SN

- Little computational power
  - Care must be used even with efficient public key cryptography and fast symmetric ciphers
- Little RAM
  - Only 4KB available
- Little bandwidth
  - Power to transmit 1 bit = power of executing 800-1000 instructions
- Little battery power
  - Each milliamp consumed = one ma closer to death
Security Issues and Focus

- SN suffer from all issues typically associated with wireless
  - Easy to eavesdrop, inject, alter data
- SN are also susceptible to:
  - Resource consumption attacks
  - Physical stealing and commandeering of motes
- TinySec only guarantees message authenticity, integrity, and confidentiality
TinySec is at the LINK-LAYER

Why Link Layer?

- End-to-end communication does not make sense for SN
  - Most communication in SN is many-to-one
  - Link-layer allows for in-network processing
- End-to-end more susceptible to DoS attacks
  - Wastes energy and bandwidth
Design Goals

- **Security**
  - Access control and Message integrity
  - Confidentiality (*Semantic security*)
  - NOT: Replay protection

- **Performance**
  - Security increases message size $\rightarrow$ less bandwidth and more power
  - Must tune strength security to minimize cost

- **Ease of Use**
  - Security Platform
  - Transparency
  - Portability
Security Primitives

- **Message Authentication Codes (MACs)**
  - Commonly used for achieving message authenticity and integrity
  - Requires a shared secret key

- **Initialization Vectors (IVs)**
  - Commonly used to achieve semantic security
  - Encrypting the same plain text twice results in different ciphertexts
  - Typically transferred in clear text
TinySec Design

- Two versions of TinySec:
  - TinySec-AE = Authentication + Encryption
  - TinySec-Auth = Authentication only

- Encryption
  - Use a cipher block chaining (CBC) encryption algorithm
    - Streams ciphers are faster, but fail if the IV is reused
    - CBC degrades less when IVs are reused
  - Use an IV with the following format:
    \[ dst|AM||l||src||ctr \]
Tweaking CBC for SN

- Since IV has a counter, there are some cases when info leakage can occur
  - Esp. when $P_1 \oplus IV = P_1' \oplus IV'$
  - Fix this by pre-encrypting the IV

- Use ciphertext stealing to avoid ciphertext expansion
  - Ensures ciphertext is same length as plain text
  - Conserves bandwidth and power
Choice of CBC algorithm

- Two options:
  - RC5
  - Skipjack

- RC5 has slightly faster implementation
- Choose Skipjack b/c it’s free and only slightly slower
TinySec Packet Format

(a) TinySec-AE packet format

(b) TinySec-Auth packet format

(c) TinyOS packet format

Figure 1: The TinySec and TinyOS packet formats. The byte size of each field is indicated below the label. Fields that have been hatched are protected by the MAC. In TinySec-AE, the data field, shaded gray, is encrypted.

- By including the dest and AM unencrypted in front, nodes can save resources through early rejection.
- TinySec-AE adds 5 bytes, TinySec-Auth adds 1 byte
Analysis of 4-byte MAC

- Traditional protocols use 8-16 byte MACs
- Too big for SN, a 4 byte MAC is good enough
  - Given a 4 byte MAC, an attacker has a 1 in $2^{32}$ chance of guessing it
  - Repeated guessing it will require $2^{31}$ tries
  - Take advantage of SN’s low bandwidth (19.2kb/s channel) here.
    - It’ll take 20 months to send $2^{31}$ packets!!
    - Batteries would run out first!!
The security of cipher block chaining (CBC) depends on unique IVs.

TinySec only uses an 8-byte IV, 4 of which change:
- Guarantees that each node can send $2^{16}$ packets before re-using an IV.
- No good for traditional networks (1Mb/s → reuse in 1 hour).
- In SN, nodes send very infrequently (GDI = 1 pkt every 70s).
  - If 1 pkt/min is sent, 45 days before IV reuse.
Keying Mechanisms

- TinySec works with a variety of keying mechanisms
  - Single network-wide shared key
  - Per-link shared key
  - Per-group shared key

<table>
<thead>
<tr>
<th>Keying mechanism</th>
<th>Benefits</th>
<th>Costs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single network-wide key</td>
<td>Simple; easy to deploy; supports passive participation and local broadcast</td>
<td>Not robust to node compromise</td>
</tr>
<tr>
<td>Per-link keys between neighboring nodes</td>
<td>Graceful degradation in the presence of compromised nodes</td>
<td>Needs a key distribution protocol; prohibits passive participation and local broadcast</td>
</tr>
<tr>
<td>Group keys</td>
<td>Graceful degradation in the presence of compromised nodes; supports passive participation and local broadcast</td>
<td>Requires key distribution; trades off robustness to node compromise for added functionality</td>
</tr>
</tbody>
</table>

Table 1: A summary of different keying mechanisms for link-layer security.
Implementation

- Portable
  - Implemented for Mica, Mica2, and Mica2Dot platforms
  - Integrated with TOSSIM
  - Ported to Texas Instruments

- Resource usage:
  - 3000 lines of NesC code
  - 728 bytes of RAM
  - 7146 bytes of ROM

- Implemented both RC5 and Skipjack

- Use upper two bits of packet length field to indicate type of packet (plain, AE, Auth)

- Globally shared secret key
TOS Scheduler Modification

- TinyOS scheduler did not provide the real-time requirements for encryption:
  - Cryptographic processing must be done by the time the radio finishes sending the start symbol
- Implemented a two-priority scheduler where cryptographic tasks run at higher priority than regular tasks
- Allows encryption and decryption to run concurrently with sending and receiving
Evaluations

- Overall:

<table>
<thead>
<tr>
<th></th>
<th>Application Data (b)</th>
<th>Packet Overhead (b)</th>
<th>Total Size (b)</th>
<th>Time to Transmit (ms)</th>
<th>Increase Over Current TinyOS Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current TinyOS Stack</td>
<td>24</td>
<td>39</td>
<td>63</td>
<td>26.2</td>
<td>—</td>
</tr>
<tr>
<td>TinySec-Auth</td>
<td>24</td>
<td>40</td>
<td>64</td>
<td>26.7</td>
<td>1.6%</td>
</tr>
<tr>
<td>TinySec-AE</td>
<td>24</td>
<td>44</td>
<td>68</td>
<td>28.3</td>
<td>7.9%</td>
</tr>
</tbody>
</table>

Table 2: Table listing the expected latency overhead incurred by TinySec. The packet overhead includes space needed for the header and media access control information, such as the start symbol. Since TinySec increases the packet size by a fixed amount, it will increase the time needed to send the packet over the radio. This impacts bandwidth, latency, and the energy needed to send a packet. We confirm this predicted overhead increase experimentally.

- Cipher performance:

<table>
<thead>
<tr>
<th>Cipher &amp; Impl.</th>
<th>Time (ms)</th>
<th>Time (byte times)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC5 (C)</td>
<td>0.90</td>
<td>2.2</td>
</tr>
<tr>
<td>Skipjack (C)</td>
<td>0.38</td>
<td>0.9</td>
</tr>
<tr>
<td>RC5 (C, assembly)</td>
<td>0.26</td>
<td>0.6</td>
</tr>
</tbody>
</table>

Table 3: Time to execute cipher operations on the Mica2 sensor nodes. We display the time both in milliseconds and in byte times.
Evaluations (Power)

Table 4: Total energy consumed to send a 24 byte packet.

<table>
<thead>
<tr>
<th></th>
<th>Energy (mAH)</th>
<th>Increase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current TinyOS Stack</td>
<td>0.000160</td>
<td>—</td>
</tr>
<tr>
<td>TinySec-Auth</td>
<td>0.000165</td>
<td>3%</td>
</tr>
<tr>
<td>TinySec-AE</td>
<td>0.000176</td>
<td>10%</td>
</tr>
</tbody>
</table>

Figure 2: The power consumption for sending a packet. All packets contained 24 byte payloads. The top graph shows the power consumption when sending the packet with the current TinyOS stack (no security). In the middle graph, we use TinySec-Auth, while the bottom graph uses TinySec-AE. Notice the large power draw at the beginning of sending as the encryption and MAC computation is overlapped with the sending of the start symbol. Additionally, note that when sending with TinySec, the packets are larger in length.

Note: Even hardware-accelerated encryption has to pay cost due to longer packets!
Evaluation (bandwidth)

Figure 3: Bandwidth, plotted as a function of the number of send-receive pairs. We compare TinySec-Auth and TinySec-AE to the bandwidth without using TinySec.

Most of the difference is due to packet size, not encryption process (see next slide).
Evaluation (packet length)

Figure 5: The increase in latency when routing packets using TinySec. We display the results in byte times, the time it takes to transmit one byte of information over the radio. This shows that TinySec’s impact on end-to-end latency is caused by the increased length of TinySec packets. There is a close correspondence between theory and practice: using authentication and encryption increases packet length by 5 bytes, and empirically, we see that latency is increased by 4.6 byte times; similarly, using authentication alone increases packet lengths by 1 byte, and empirically it increases latency by 1.1 byte times. Note that we have normalized by the route distance.
Evaluation (Ease of Use)

- Integrated TinySec with a large-scale test application
  - Did NOT require changing any application code
  - Only required changing one line in the makefile
  - This is expected to be the general case

- There are already a bunch of other users using TinySec
  - TinyPK – RSA to exchange TinySec keys
  - TinyCrypt – ECC to exchange TinySec keys
  - RSI and Bosch companies both use it
Conclusion

- TinySec is tailored to SNs
- Relies on existing encryption schemes
- Adds minimal overhead
- Already in wide use throughout SN community
- Extensively evaluated